Noname manuscript No.
(will be inserted by the editor)

Solving the Shifts and Breaks Design Problem Using Integer Linear Programming

Arjan Akkermans · Gerhard Post · Marc Uetz

Received: date / Accepted: date

Abstract In this paper we propose a two-phase approach to the shifts and breaks design problem using integer linear programming. In the first phase we create the shifts, while heuristically taking the breaks into account. In the second phase we assign breaks to each occurrence of any shift, one by one, repeating this until no improvement is found. This approach outperforms the current best known method for shifts and breaks design on a set of benchmark instances, as well as on real life instances.

Keywords Shift Design · Break Scheduling · Timetabling

1 Introduction

Personnel scheduling was first introduced by Edie [15] and formulated as a set covering problem by Dantzig [10] in the 1950's. After its introduction it has received a great deal of attention in the literature and has been applied to numerous areas such as airlines, health care systems, police, call centres and retail stores [16]. The interest can be explained by labour cost being a major direct cost component for companies.

In this paper we consider the shifts and breaks design problem which in this specific form was introduced by Di Gaspero et al [14]. In this problem a set of shifts has to be selected from a set of possible shift types. A shift is characterised by its starting time and length, such as 09:00 and 8 hours. A selected shift can then be used on multiple days and with a different number of *duties* assigned to it on each day. A duty represents shift with a person assigned to it, and corresponds to a workload to be carried out by a single person. One of the goals in the shifts design problem is to select a small number of different shifts. This because schedules with a small number of different shifts are easier from a managerial perspective. Furthermore the number of duties over the planning horizon must follow a given *staffing requirement* at any time, which is given as a piecewise constant function that encodes the number of required duties per unit time. Here we understand 'to follow' in the sense that under- and overstaffing is possible yet undesirable. Eventually this will translate into penalty costs. Figure 1 and Table 1 show, respectively, an example of staffing requirements over a two-day period, and

Arjan Akkermans · Gerhard Post · Marc Uetz
Department of Applied Mathematics
University of Twente
P.O. Box 217, Enschede
The Netherlands

Gerhard Post
ORTEC Optimization Technology
Houtsingel 5, Zoetermeer
The Netherlands

Fig. 1: Example Staffing Requirements

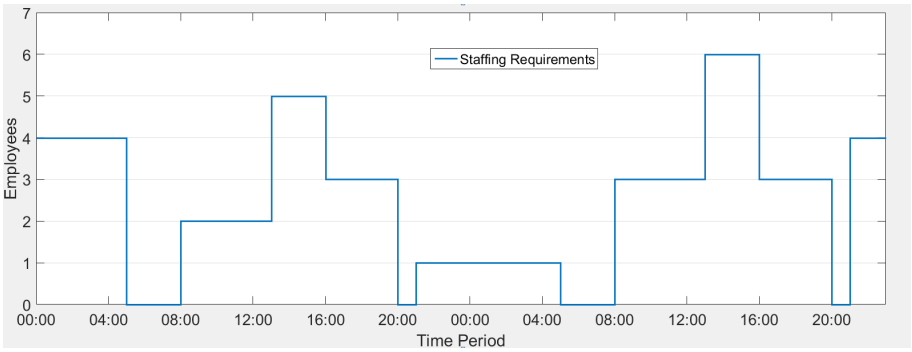


Table 1: Example Shift Types

Abbr.	Name	Earliest start	Latest start	Minimum length	Maximum length
M	Morning Shift	05:00	08:00	7:00	9:00
D	Day Shift	09:00	11:00	7:00	9:00
E	Evening Shift	13:00	15:00	7:00	9:00
N	Night Shift	21:00	23:00	7:00	9:00

a set of possible shift types. The problem is *cyclic* hence duties working on a night shift on the last day will continue to be working on the first time periods of the next planning horizon.

On top of the selected shift design, for each duty an individual break allocation has to be scheduled. Breaks are restricted by a large number of (legal) regulations and conditions which we will discuss in more detail later. Roughly said, frequent small breaks are required in the available instances. Obviously, duties are not counted as contributing to the fulfilment of the staffing requirements during a break, and in fact, also not during the time period immediately after each break.

2 Problem Description

We use the same problem formulation as Di Gaspero et al [14]. The planning horizon consists of a set of consecutive days such as a week. The planning horizon is divided into time periods of a constant length such as 5 minutes. For each time period t the staffing requirement R_t is given. A solution is represented by a set S of selected shifts, the number of duties assigned to each shift on each day, and a break allocation to each duty.

Shifts must be chosen in accordance with pre-defined shift types. For each shift type the minimum and maximum values for the starting time and the duration for a shift of the shift type are given. Once a shift is chosen, the number of duties assigned to each shift on each day has to be determined.

To each duty an individual break allocation must be assigned, which splits a shift into a set of alternating *break* and *working periods*. Clearly, a duty that is on break at a certain time period, does not contribute to the staffing requirements. The same holds for the period following a break: the reasoning is that the employee needs some startup time, which makes him not effective in this period. The periods of the duty that contribute to the staffing requirements are called *active*.

Various restrictions are set on the break allocation. These represent legal regulations as well as physical requirements that prevent fatigue. Let us briefly discuss these constrains: Firstly, the total break time of a duty is fixed. It depends on the length of the duty. The number of breaks of a duty is not fixed, however. Furthermore all breaks and working periods have a minimum and maximum length. For example, if a break follows a *long* working period, which is a working period longer than a certain threshold, then the next break has a longer

minimum length. Also, the start of a break cannot be too close to the beginning and end of a shift. Finally, a duty of sufficient length requires a lunch break of a fixed length which has to be scheduled around the middle of the duty.

The objective function is the weighted sum of overstaffing, understaffing, and the number of selected shifts. The first two factors can easily be defined by using the active time periods of a duty. Let a_t to represent number of duties that are active at time period t. The overstaffing, O, represents the total excess of active duties and the understaffing, U, the total shortage. They are defined as follows

$$O = \sum_{t \in T} \max\{a_t - R_t, 0\}, \quad U = \sum_{t \in T} \max\{R_t - a_t, 0\}$$

Using the weights W_1, W_2 and W_3 to penalise the different solution criteria, and using $|S|$ to represent the total number of selected shifts, the objective is formulated in the following way.

$$\text{minimize } W_1 \cdot O + W_2 \cdot U + W_3 \cdot |S| \tag{1}$$

3 Previous and Related Work

After the personnel scheduling problem was introduced by Dantzig [10], many different formulations of personnel scheduling problems have been considered. A recent overview of personnel scheduling is given by Van den Bergh et al [7]. The authors note that a wide range of solution methods is used to find solutions to personnel scheduling problems. The two main approaches used are mathematical programming methods such as linear programming, goal programming, integer programming and column generation, and heuristic methods such as simulated annealing, tabu search and genetic algorithms. Other approaches include simulation, constraint programming and queueing.

Efficient methods for solving personnel scheduling problems without breaks are available for some specific formulations. The constraint matrix in the integer linear programming formulation of Dantzig [10], that describes a set covering problem, is a consecutive ones matrix if the problem does not contains cyclicity nor breaks, i.e. all duties are active on a consecutive number of time slots. Matrices with the consecutive ones property are totally unimodular, as first shown by Veinott and Wagner [29], and hence integer linear programs containing such a constraint matrix can be solved efficiently by solving the linear programming relaxation. Bartholdi et al [4] show that personnel scheduling problems with a row circular matrix can be solved efficiently by considering at most a bounded number of flow problems. These row circular matrices follow if all duties are of the same length and contain no breaks. Furthermore Bartholdi et al [4] show that the formulation of Dantzig [10] can be slightly changed to allow for problems in which, next to the linear cost for using a duty, a linear penalty cost for overstaffing and understaffing can be considered for which the efficient solution methods as discussed still hold.

Even when ignoring the problem of minimizing the number of shifts, the constraint matrix for the shift design problem has neither the consecutive ones property, nor the circular row property. The constraint matrix is column circular. Cyclic scheduling problems involving duties without breaks will always have the circular ones property in the columns. Hochbaum and Levin [18] discuss the hardness of problems with a column circular matrix, and show the problem to be equivalent to the exact matching problem which is in the complexity class NRC [25]. It is currently unknown whether the problem is in P.

Aykin [3] studied a problem for a continuous 24 hour workday (cyclical) and giving each duty one half hour lunch break and two smaller 15 minute breaks. For this problem an integer linear program formulation is described which differs from the original set cover formulation proposed by Dantzig [10] by requiring three breaks to be scheduled. Rekik et al [28] considered an extension of the model and tested instances for which exactly three breaks were required for each duty, having a combined duration of two hours. Furthermore, the second break was required to be the longest break and the continuous periods in-between two breaks (working

period) was restricted to have a length between one and three hours. For this problem Rekik et al [28] propose two integer linear program formulations and compare their results with modified versions of integer linear programming formulations given by Bechtold and Jacobs [5] and Aykin [3]. Rekik et al [28] show that their model is slightly slower computationally than the formulations in [5] and [3], which is explained by the added flexibility that their model gives.

The shifts and breaks design problem of the form discussed here was introduced in [14]. The authors state that to the best of their knowledge the very problem as described in their paper has not been addressed before in the literature. The authors propose an approach combining local search (LS) and constraint programming (CP) [2]. Their approach starts off with finding an initial randomly generated solution for the assignment of shifts. Random LS is employed on a part of the solution, namely a solution which specifies the shifts, the number of duties for each shift on each day, and the number of breaks for each duty. A CP model is then used to find a feasible break allocation. Since the authors are the first to tackle the shifts and breaks design problem there are no results to compare to. The authors use a set of randomly generated instances as well as a set of real-life instances which are publicly available [11].

The shifts and breaks design problem is the combination of two other problems: the shifts design problem [26] and the break scheduling problem [6]. Local Search techniques have been applied to both of these problems. Musliu et al [26] use tabu search [17] to guide their local search for the shift design problem. The authors state that they rely on local search techniques because the shift design problem is proven to be NP hard [21]. Di Gaspero et al [13] use a hybrid heuristic, which consists of a greedy heuristic followed by a local search algorithm, which outperforms the previous results. The greedy heuristic is based on the relation of the shift design problem to a flow problem as described in [21].

Answer Set Programming (ASP) [9] is an exact solution technique and has been applied to the shift design problem by Brewka et al [9]. ASP is able to find most best known solutions within 60 minutes on instances of shift design where those solutions have no overstaffing and understaffing. While the execution times are often not competitive with results from [13], there are instances on which ASP works very well. Solutions found for the instances in which a solution without overstaffing and understaffing might not exist are not competitive with results from the literature. The authors state that a combination of ASP together with domain-specific heuristics is a promising area for further research.

A variation on the shift design problem is studied by Bonutti et al [8]. The authors extend the shift design problem to have multiple types of skills for employees. The requirements state the required number of employees at each time period for each skill. This formulation also allows for the possibility of planning one break around the middle of a shift. Simulated annealing [20] is used to guide the search process.

Break scheduling was first introduced by Beer et al [6]. In their formulation different constraints are given with different weights to each type of violation and there are no hard constraints. Their formulation uses either a random assignment of breaks as an initial solution or a solution following from a simple temporal problem [12]. After the initial solution has been constructed the authors use local search guided by either tabu search, simulated-annealing, or a minimum conflicts-based heuristic [23]. A memetic algorithm for break scheduling was proposed by Musliu et al [27]. Memetic algorithms combine population based method with local search techniques and were first mentioned in [24]. The authors compare their results with the minimum conflicts-based heuristic as proposed by Beer et al [6]. The algorithms both score best on half of the test instances and hence conclusions regarding the better algorithm are indecisive. An improved memetic algorithm is given in [30]. This algorithm obtains best results on all the instances considered by the previous authors. In this approach a part of the constraints in the break scheduling problem are considered hard and, ignoring the other constraints which regard the staffing requirements, a simple temporal problem [12] can be formulated. Solutions from this are taken as initial solutions for the memetic algorithm. In this approach memes are defined by a set of time slots and each duty is assigned to exactly one meme in which most of the timeslots of the duty are.

The reason for this approach as opposed to the approach used in [27] is that duties have a strong interference in satisfying the staffing requirements, and therefore it is difficult to find effective crossover operations when a meme represents a duty. This memetic algorithm is also shown in [31]. In this paper it is also proven that break scheduling is generally NP-complete, even under the condition that all feasible break patterns of each duty are given explicitly in the input.

4 The Two-Phase ILP Approach

Solving the shifts and breaks design problem in one stroke is computationally hard, especially because of the importance to minimize the number of active shifts. For this reason we propose a solution method that is composed of two stages: the first stage determines which shifts and how many duties will be used, while in the second stage the breaks are assigned to the duties.

More concretely, our solution approach to the shifts and breaks design problem consists of a two-phase approach where we use integer linear programming (ILP) to find a solution in each phase. In the first phase the shifts are designed and the number of duties for each shift and day are determined. The problem that we solve in the first phase is a variation of shift design in which we have to select a set of shifts while accounting for break time which has to be allocated in the second phase. This second phase will treated as a series of instances of the break scheduling problem.

4.1 The First Phase Method

First we give an integer linear program (ILP) formulation of the shift design problem. A compact overview of this ILP is given in A.2. The first phase will use a modified version of this ILP which we will discuss after having introduced the shift design ILP.

4.1.1 The ILP for the Shift Design Problem

The shift design problem is made up of time periods, days and shifts. We will use the following variables to refer to a single element of each set.

- t refers to a single time period.
- d refers to a single day.
- s refers to a single shift.

The time periods and the days are part of the input, but the shifts are not explicitly given. However, we can use the shift types to determine all possible shifts by noting that all unique shifts of a certain type can be found by using all unique pairs of starting times and lengths of that shift type. For each possible shift we are then making the decision of using it or not. In the case that a shift is used, we call it *active* and we can assign a number of duties to it on each day. Hereto we use the following variables.

$$a_s = \begin{cases} 1 & \text{if shift } s \text{ is active} \\ 0 & \text{otherwise} \end{cases}$$

$$w_{d,s} = \text{ the number of duties of shift } s \text{ starting on day } d$$

If a shift is inactive the number of duties on each day are forced to be 0. To model this constraint we use the parameter $M = max_t\{R_t\}$. Recall that R_t denotes the staffing requirements at time period t. M is an upper bound on the number of duties which are used in an optimal solution for shift design since using more duties will lead to unnecessary overstaffing. The following constraints force the number of duties to be 0 in case the corresponding shift is not active.

$$w_{d,s} \leq M \cdot a_s \quad \forall d, s \tag{2}$$

Using the number of duties for each shift we can can calculate the overstaffing and the understaffing. For this we need a parameter denoting on which time periods the duties of a shift starting on a specific day are active. Recall that breaks are not part of the shift design problem. We introduce the following.

$$o_t = \text{ the amount of overstaffing at time period } t, \quad o_t \geq 0$$
$$u_t = \text{ the amount of understaffing at time period } t, \quad u_t \geq 0$$
$$A_{s,d,t} = \begin{cases} 1 \text{ if the duties of shift } s \text{ starting on day } d \text{ are active on time period } t \\ 0 \text{ otherwise} \end{cases}$$

The following constraints specify, respectively, lower bounds for the understaffing and the overstaffing variables.

$$\sum_{s,d} \left(A_{s,d,t} \cdot w_{d,s}\right) + u_t \geq R_t \qquad \forall t \tag{3}$$

$$o_t = \sum_{s,d} \left(A_{s,d,t} \cdot w_{d,s}\right) + u_t - R_t \qquad \forall t \tag{4}$$

The objective function is the weighted sum of overstaffing, understaffing and the number of shifts.

$$\text{minimize } W_1 \sum_t o_t + W_2 \sum_t u_t + W_3 \sum_s a_s \tag{5}$$

4.1.2 Virtual Shifts

In order to design shifts that follow the requirements in the shifts and breaks design problem more closely, we need to account for the break time which has to be allocated to each duty in the second phase. In order to do this we construct a *virtual* shift for each possible shift. These are used to represent a shift while heuristically accounting for the break time which has to be scheduled. Instead of a binary indicator for denoting whether a duty will be active at a time period, the virtual shifts can be thought of as representing the probability of a duty being active at a time period.

In the shift design ILP we used the parameters $A_{s,d,t}$, which were binary and indicated whether duties of shift s starting on day d were active during time period t. For the virtual shifts we will use the values $A^*_{s,d,t}$ which can be any value from 0 to 1. A duty is not active on their breaks and on the first time period following a break. The total break time required for a shift is given as part of the input by $f(s)$. Before assigning the breaks we do not know the number of breaks that a duty will have and hence we can not determine the number of inactive time periods of that duty exactly. However, we can calculate the maximum and minimum number of breaks of a duty by using the various restrictions on the break allocations. We will use $IP(s)$ to indicate the number of inactive time periods for a duty belonging to shift s, under the assumption of having as many breaks as possible. Hereto we use $maxB(s)$ to denote the maximum number of breaks for shift s, which leads to

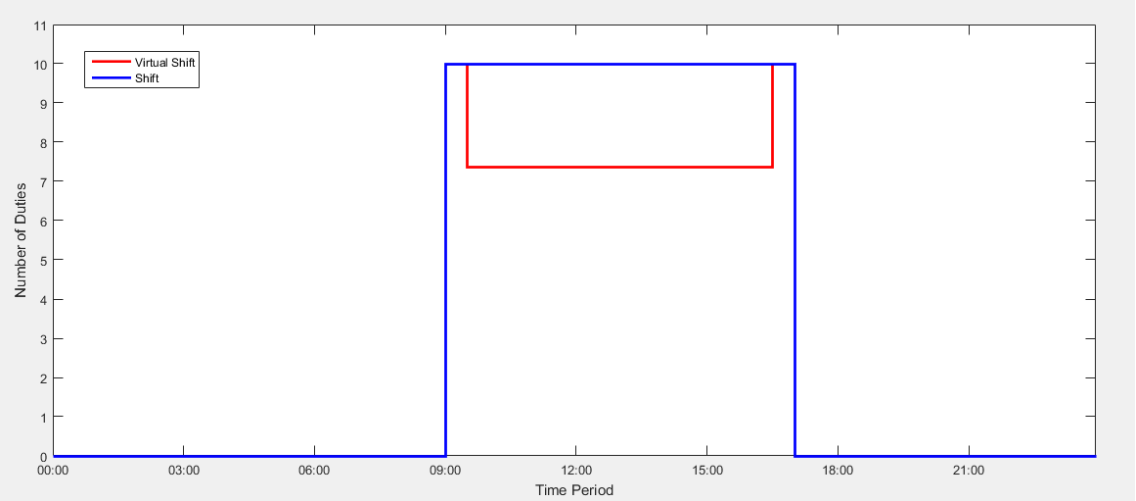
$$IP(s) = f(s) + maxB(s) \tag{6}$$

We use the maximum number of breaks to account for the worst case. Therefore this approach is conservative, and may lead to scheduling too many duties. This is justified, however, since in most staff scheduling problems, understaffing is considered to be less desirable than overstaffing.

Recall that breaks cannot be scheduled near the beginning and end of a shift. Denote by α and β the number of time periods where the shift is guaranteed to be active at the start and at the end of the shift, respectively. Outside these time intervals a duty might be inactive depending on the break schedule. We simply 'divide' the rest of the breaks evenly over the remaining time periods of the duty. We use R^*_s to denote the ratio of inactive periods to active periods of shift s, excluding the first α and last β time periods, so

$$R^*_s = \frac{IP(s)}{s.length - \alpha - \beta} \tag{7}$$

Fig. 2: Shift and Virtual Shift



That said, we define the parameters $A^*_{s,d,t}$, which can be interpreted as the probability of a duty being inactive on time period t, as follows

$$A^*_{s,d,t} = \begin{cases} 0 & \text{if } t \text{ is not part of the duties of shift } s \text{ starting on day } d \\ 1 & \text{if } t \text{ is in the first } \alpha \text{ time periods of the duties of shift } s \text{ starting on day } d \\ 1 & \text{if } t \text{ is in the last } \beta \text{ time periods of the duties of shift } s \text{ starting on day } d \\ 1 - R^*_s & \text{otherwise} \end{cases} \tag{8}$$

Figure 2 shows the effect of working with a virtual shift (with 10 duties).

4.1.3 The ILP for the First Phase

The shift design ILP can be adapted in various ways to incorporate the virtual shifts. It is possible to simply change the parameters $A_{s,d,t}$ to $A^*_{s,d,t}$. Preliminary tests showed that this approach gives decent results, however a drawback is that the actual flexibility of being able to choose the break allocation is not taken into account. Intuitively, simply working with virtual instead of actual shifts may be overly pessimistic.

With an eye on the fact that understaffing is generally less desirable than overstaffing, we work with virtual shifts in a slightly 'asymmetric' way, by adapting the constraints that express under- and overstaffing.
For understaffing we simply use the constraint (3):

$$\sum_{s,d} \left(A_{s,d,t} \cdot w_{d,s}\right) + u_t \geq R_t \quad \forall t \tag{9}$$

In other words, we count understaffing only in the case that there are not enough duties even without the breaks. In such a case understaffing would indeed be inevitable.

Overstaffing at a time period, however, will be penalized only if there are too many duties when using the virtual values $A^*_{s,d,t}$ as defined by the virtual shifts.

$$o_t \geq \sum_{s,d} \left(A^*_{s,d,t} \cdot w_{d,s}\right) + u_t - R_t \qquad \forall t \tag{10}$$

Therefore, we only count overstaffing if there are too many duties using the virtual shifts. This updated ILP accepts without penalty all overstaffing and understaffing, as long as the number of present workers is between the number specified by the shifts and the virtual shifts.

Using only these two constraints, it is possible that overall too few duties are scheduled since the the constraints allow a duty to be always active. To tackle this issue, we enforce that over a certain number C of consecutive time periods the sum of the staffing requirements are fulfilled using the virtual shifts. Note that we abuse notation by allowing the index $t+i$ to be greater than the number of time periods (n). In this case $t+i$ should be read as $(t+i)$ **mod** n.

$$\sum_{i=1}^{C} \left(\sum_{s,d} A^*_{s,d,t+i} \cdot w_{d,s} \right) \geq \sum_{i=1}^{C} R_{t+i} \quad \forall t \tag{11}$$

As this equation is a hard constraint, we chose C experimentally and rather large. Our experiments suggested to work with the value $C = 25$, expressing the idea that understaffing at a certain time period can be resolved by borrowing overstaffing from 'adjacent' time periods.

4.2 Second Phase Method

Here we give a description of the algorithm used to allocate the breaks. This algorithm uses a break scheduling ILP which considers the break allocation for a single duty at a time and takes the other break allocations as fixed. This ILP is given in Appendix A.3. For an extensive description of this ILP we refer to [1].

The algorithm for the second phase is a greedy algorithm. The duties on each day as determined by the first phase are used to determine the initial number of active duties at each time period, where at the start of phase two, all duties are active from the start to the end of the duty (i.e., no breaks are scheduled). Then the duties are considered in a fixed but random order, and a break allocation is calculated for each duty consecutively by using the break scheduling ILP. Note that this alters the number of active duties at some of the time periods, and hence the objective function. This greedy procedure continues as long as it is possible to improve the solution by changing the break allocation of a single duty, and terminates at a local optimum.

5 Computational Results

The website [11] contains a set of instances that was used to test our method. The instances consist of two sets, referred to as 'First Set' and 'Second Set' in Tables 2 and 3. Both sets contain 30 randomly generated instances. These instances were created by 'reverse engineering' as follows: The staffing requirements are generated from a solution with a randomly generated set of active shifts and duties. Hence, there is a 'best known' solution which provides an exact coverage of the requirements.

The third set of 5 instances is based on a real world example. For these instances a good solution in which there is no overstaffing and understaffing is very unlikely to exist. The real life instances are smaller than the randomly generated instances with respect to the total staffing requirements. For the real life case the average value for the sum of the staffing requirements over all time periods equals $10,193$, for the randomly generated instances this number is $16,535$. However, the given shift types for the real life instances allow more shifts. For these instances the total number of possible shifts is 8645, while the randomly generated instances allow for 2800 possible shifts. Due to the higher number of possible shifts for the real life instances our method did not satisfactory results when solving an ILP containing all possible shifts for these instances. To overcome this challenge we only considered shifts at a time granularity of 15 minutes for the real life instances. This reduces the number of possible shifts from 8645 to 1075 shifts, but might cut out the optimal solution.

Results for the randomly generated instances are shown in the Tables 2 and 3. For each instance we display the objective value of the two-phase approach and compare it to the results of Di Gaspero et al [14]. Next we show

the best known solution which was used to construct the instances. We also list the time taken (in minutes) by our two-phase approach. Results of the real life instances are shown in Table 4. For these instances we included the values for overstaffing, understaffing and the number of shifts. The 'LB' column shows a lower bound to *the shifts and breaks problem* using the shifts and number of duties provided by the first phase. A lower bound for the best possible objective value can be obtained by scheduling as many breaks as possible at time periods on which there is overstaffing. Therefore in the best case scenario, the penalty function for the number of shifts is exactly known since the first phase has already designed the shifts. For understaffing, the best case is that no additional understaffing is created by scheduling the breaks. For this penalty value we only take into account *unavoidable* understaffing which is the total understaffing in case that no breaks are scheduled. For the overstaffing, the best case would be when a maximum number of breaks is scheduled for each duty (to get rid of as much overstaffing as possible). The penalty value for overstaffing is the difference between sum of total active duties with as many breaks as possible, and the sum of the requirements. The tables show that in the randomly generated instances, the lower bound is on average 5.1% better when compared to the objective value found by our two-phase approach. For the real life instances this is 5.7%. Di Gaspero et al [14] use a time limit of 60 minutes per instance to find their solutions. We allow a time limit of 30 minutes for the first phase and a time limit of 30 minutes for the second phase. For the first phase these 30 minutes are always used. The average optimality gap in the first phase for the randomly generated instances is 0.50 (with standard deviation 0.10), for the real life instances this number is 0.86 (with standard deviation 0.02). A time limit of 30 minutes was also used for the second phase. Therefore a running time of lower than 60 minutes means that the second phase converges before the allocated 30 minutes are used, this is the case in 40 of the randomly generated instances and for 3 of the real life instances.

The algorithm for shifts and breaks design was written in C++ using Microsoft Visual Studio 2013 [22]. The integer linear programs were solved using the commercially available optimisation package Cplex [19]. The algorithm was performed on a PC with an Intel i7-4702MQ quad core processor with 2.2Ghz and 8GB RAM memory.

Table 2: Results First Set

Inst.	Objective				Time
	LB	2P-ILP	Hybrid	Best	
1-1	3176	3284	10540	480	42
1-2	5226	5360	14904	600	40
1-3	2960	3108	15330	600	48
1-4	7066	7410	18652	960	56
1-5	4260	4494	11656	480	42
1-6	2726	2830	8756	420	37
1-7	4662	4872	10042	540	48
1-8	5276	5450	14210	600	53
1-9	4236	4440	12120	600	51
1-10	5902	6150	15804	660	**60**
1-11	2060	2076	n.a.[1]	120	39
1-12	2776	2944	8360	360	51
1-13	3738	3898	12306	420	**60**
1-14	6464	6760	18146	780	**60**
1-15	2520	2548	4774	180	34
1-16	6220	6506	15820	900	48
1-17	7006	7348	18402	1080	59
1-18	5130	5428	16668	720	58
1-19	4408	4772	13582	720	**60**
1-20	4582	4838	16794	540	**60**
1-21	4286	4396	10188	480	43
1-22	1872	1944	9816	300	37
1-23	5632	5822	13626	600	45
1-24	4208	4456	11730	480	42
1-25	5896	6254	18436	960	54
1-26	5440	5644	16286	660	48
1-27	3960	4362	18484	480	**60**
1-28	4256	4398	9952	540	40
1-29	5720	5906	13646	720	51
1-30	3352	3368	8604	300	39
Av.	4501	4702	13367	576	49

Table 3: Results Second Set

Inst.	Objective				Time
	LB	2P-ILP	Hybrid	Best	
2-1	5064	5380	14002	720	52
2-2	5420	5594	12866	720	**60**
2-3	5944	6694	13858	720	**60**
2-4	5088	5282	12780	720	58
2-5	5144	5400	12962	720	**60**
2-6	5822	6696	16214	720	**60**
2-7	6362	6816	17044	720	49
2-8	5606	5756	13684	720	44
2-9	5844	6110	14932	720	47
2-10	5878	6206	17972	720	58
2-11	6312	6648	n.a.[1]	960	48
2-12	6816	7082	16028	960	52
2-13	6638	6908	17446	960	57
2-14	6268	6552	18636	960	**60**
2-15	6794	7268	19032	960	**60**
2-16	6810	7370	18950	960	**60**
2-17	6700	6944	15754	960	53
2-18	6516	7022	18616	960	**60**
2-19	6660	6980	19456	960	**60**
2-20	6782	7256	18688	960	54
2-21	8354	8804	18890	1200	**60**
2-22	6928	7328	19804	1200	58
2-23	7818	8390	17236	1200	58
2-24	6810	7228	18178	1200	59
2-25	6974	7428	19198	1200	**60**
2-26	7514	8046	19662	1200	**60**
2-27	8076	8948	20200	1200	**60**
2-28	7584	7816	16414	1200	**60**
2-29	7558	8530	18574	1200	**60**
2-30	7658	8012	24462	1200	58
Av.	6591	7016	17294	960	57

Table 4: Results Real Life Instances

Instance	Overstaffing		Understaffing		Shifts		Objective			Time
	Hybrid	2P-ILP	Hybrid	2P-ILP	Hybrid	2P-ILP	Hybrid	2P-ILP	LB	2P-ILP
2fc04a	2636	1224	173	5	23	30	8382	4298	4050	56
3fc04a	2732	1227	130	1	21	36	8024	4624	4398	57
4fc04a	2710	1081	94	4	21	32	7620	4122	3896	56
50fc04	2636	1130	180	4	29	29	8812	4040	3744	**60**
51fc04	2890	1343	209	0	23	32	9250	4606	4360	**60**
Average	2720	1201	157	3	23	32	8418	4338	4090	58

[1] The authors did not report a solution

6 Conclusion

We proposed an integer linear programming approach for the shifts and breaks design problem, which obtains better results on all instances when compared to the best results available in the literature. Note that the integer linear program of the first phase can also be used to solve the instances of shift design problem, which are all solved to optimality within the time limit of 30 minutes. These result are shown in Appendix A.1.

Since we were not able to solve the ILP in the first phase to optimality we are not sure of the performance of our algorithm in case the optimal solution for this ILP would have been be found. For the randomly generated instances, for which a solution that exactly matches the staff requirements is known, the average objective value of our solutions are more than 5 times higher than the cost of the solution that exactly match the requirements. Based on the solution of the first phase, we can estimate a lower bound for the final solution, see the 'LB' column in the Tables 2, 3, and 4. This actually suggests that phase two can hardly be improved. In other words, if we want to improve the two-phase approach, it should probably be the first phase.

The low understaffing is partly a consequence of the choice made in Constraint (11) where we did not allow any understaffing over 25 time periods as measured by the virtual shifts. Although this requirement seems rather weak, it creates solutions with almost no understaffing. We experimented with soft variants of Constraint (11), which led to inconclusive results. The main reason seems to be that obtaining good lower bounds for the problem is more difficult, leading to increased computation times, and worse solutions on several instances, as well as large improvements on other ones.

As argued above, we observe that the second phase works practically very well in the two-phase approach that we propose. However, applying phase two solely on instances of break scheduling turned out to be less effective than the memetic algorithms of [30], when starting with the shifts and duties that were used to construct the instances.

References

1. Akkermans A (2017) A two-phase approach to the shifts and breaks design problem using integer linear programming. Master's thesis, University of Twente, URL http://essay.utwente.nl/74147/
2. Apt K (2003) Principles of Constraint Programming. Cambridge University Press, New York, NY, USA
3. Aykin T (1996) Optimal shift scheduling with multiple break windows. Management Science 42(4):591–602
4. Bartholdi JJ, Orlin JB, Ratliff HD (1980) Cyclic scheduling via integer programs with circular ones. Operations Research 28(5):1074–1085
5. Bechtold SE, Jacobs LW (1990) Implicit modeling of flexible break assignments in optimal shift scheduling. Management Science 36(11):1339–1351
6. Beer A, Gartner J, Musliu N, Schafhauser W, Slany W (2010) An AI-based break-scheduling system for supervisory personnel. IEEE Intelligent Systems 25(2):60–73
7. Van den Bergh J, Beliën J, De Bruecker P, Demeulemeester E, De Boeck L (2013) Personnel scheduling: A literature review. European Journal of Operational Research 226(3):367 – 385
8. Bonutti A, Ceschia S, De Cesco F, Musliu N, Schaerf A (2016) Modeling and solving a real-life multi-skill shift design problem. Annals of Operations Research 252:365–382
9. Brewka G, Eiter T, Truszczynski M (2011) Answer set programming at a glance. Commun ACM 54(12):92–103
10. Dantzig GB (1954) A comment on Edie's "Traffic Delays at Toll Booths". Journal of the Operations Research Society of America 2(3):339–341
11. Database and Artificial Intelligence Group, Vienna University of Technology (2017) Shift design and break scheduling benchmarks. URL http://www.dbai.tuwien.ac.at/proj/SoftNet/Supervision/Benchmarks/, accessed: 04-04-2017
12. Dechter R, Meiri I, Pearl J (1991) Temporal constraint networks. Artificial Intelligence 49(1):61 – 95
13. Di Gaspero L, Gärtner J, Kortsarz G, Musliu N, Schaerf A, Slany W (2007) The minimum shift design problem. Annals of Operations Research 155(1):79–105

14. Di Gaspero L, Gärtner J, Musliu N, Schaerf A, Schafhauser W, Slany W (2010) A hybrid LS-CP solver for the shifts and breaks design problem. In: 7th International Workshop on Hybrid Metaheuristics, Springer, Heidelberg, Lecture Notes in Computer Science, vol 6373, pp 46–61
15. Edie LC (1954) Traffic delays at toll booths. Journal of the operations research society of America 2(2):107–138
16. Ernst A, Jiang H, Krishnamoorthy M, Sier D (2004) Staff scheduling and rostering: A review of applications, methods and models. European Journal of Operational Research 153(1):3 – 27
17. Glover F, Laguna M (1999) Tabu search. In: Du DZ, Pardalos PM (eds) Handbook of Combinatorial Optimization: Volume1–3, Springer US, Boston, MA, pp 2093–2229
18. Hochbaum DS, Levin A (2006) Cyclical scheduling and multi-shift scheduling: Complexity and approximation algorithms. Discrete Optimization 3(4):327 – 340
19. IBM (2017) IBM ILOG CPLEX Optimization Studio 12.7.1
20. Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220(4598):671–680
21. Kortsarz G, Slany W (2001) The minimum shift design problem and its relation to the minimum edge-cost flow problem. Tech. Rep. DBAI-TR-2001-46, Technische Universität Wien
22. Microsoft (1980) Microsoft Visual Studio Community 2013, Version 12.031101.00 Update 4
23. Minton S, Johnston MD, Philips AB, Laird P (1992) Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. Artificial Intelligence 58(1):161 – 205
24. Moscato P (1989) On evolution, search, optimization, genetic algorithms and martial arts - towards memetic algorithms. Technical Report, California Institute of Technology
25. Mulmuley K, Vazirani U, Vazirani V (1987) Matching is as easy as matrix inversion. Combinatorica 7(1):105–113
26. Musliu N, Schaerf A, Slany W (2004) Local search for shift design. European Journal of Operational Research 153(1):51 – 64
27. Musliu N, Schafhauser W, Widl M (2009) A memetic algorithm for a break scheduling problem. In: Proc. 8th Metaheuristic International Conference (MIC 2009), Hamburg, Germany
28. Rekik M, Cordeau JF, Soumis F (2010) Implicit shift scheduling with multiple breaks and work stretch duration restrictions. Journal of Scheduling 13:49–75
29. Veinott AF, Wagner HM (1962) Optimal capacity scheduling-i. Operations Research 10(4):518–532
30. Widl M, Musliu N (2010) An improved memetic algorithm for break scheduling. In: Blesa MJ, Blum C, Raidl G, Roli A, Sampels M (eds) Hybrid Metaheuristics: HM 2010, Lecture Notes in Computer Science, vol 6373, Springer, Berlin, pp 133–147
31. Widl M, Musliu N (2014) The break scheduling problem: complexity results and practical algorithms. Memetic Computing 6(2):97–112

Appendix

A.1 Results Shift Design

Comparison of ILP formulation of shift design problems solved with the commercial solver Cplex[19] and the greedy min-cost max-flow + Local Search heuristic (Gr) proposed by Di Gaspero et al [13]. Table 5 show the objective of the (previous) best known solution and running time of 'Gr' to reach it. For our ILP approach we show the optimal objective value and the running time to find and prove it. Table 6 shows the results of both approaches using a running time of 1 second.

Table 5: Shift Design Comparison Set 1

Inst.	Objective		Time (Seconds)	
	Gr	ILP	Gr	ILP
1-1	480	320	1	2
1-2	300	212	40	21
1-3	600	374	2	2
1-4	450	340	109	165
1-5	480	319	2	2
1-6	420	213	1	1
1-7	270	237	7	1
1-8	150	147	11	149
1-9	150	149	9	29
1-10	330	289	84	141
1-11	30	30	1	1
1-12	90	81	4	3
1-13	105	105	4	9
1-14	195	187	61	425
1-15	180	170	0	1
1-16	225	209	152	1552
1-17	540	394	288	283
1-18	720	447	7	6
1-19	180	177	31	54
1-20	540	353	2	2
1-21	120	119	2	7
1-22	75	75	4	4
1-23	150	150	22	100
1-24	480	343	1	6
1-25	480	352	n.a.[2]	168
1-26	600	347	9	5
1-27	480	393	2	3
1-28	270	222	4	32
1-29	360	289	10	58
1-30	75	75	2	2
Average	318	237	30	108

Table 6: Shift Design Comparison Set 3

Inst.	Objective	
	Gr	ILP
3-1	2386	318
3-2	7691	845
3-3	9597	924
3-4	6681	1427
3-5	9996	551
3-6	2077	1892
3-7	6087	642
3-8	8861	725
3-9	6036	2527
3-10	3002	462
3-11	5491	1024
3-12	4171	3514
3-13	4662	3131
3-14	9661	701
3-15	11445	1112
3-16	10734	638
3-17	4729	3011
3-18	6692	893
3-19	5157	2677
3-20	9175	1845
3-21	6054	4674
3-22	12870	2063
3-23	8390	699
3-24	10418	741
3-25	13252	847
3-26	13118	1042
3-27	10081	1034
3-28	10604	887
3-29	6690	1045
3-30	13724	1011
Average	7984	1430

[2] Authors were not able to find the best known solution using the Gr algorithm

A.2 ILP Shift Design

Sets	
Time Periods	t
days	d
shifts	s
Variables	
a_s	Binary variable indicating whether shift s is active
$w_{d,s}$	Integer variable indicating how many workers will be working on shift s on day d
u_t	Will denote the understaffing at time period t
o_t	Will denote the overstaffing at time period t
Parameters	
R_t	The requirement at time period t
$A_{s,d,t}$	Binary, indicating if the duty starting on day d of shift s is active on time period t
W_1	Penalty cost for overstaffing
W_2	Penalty cost for understaffing
W_3	Penalty cost for the number of shifts
M	Is used as a large constant, here it is the maximum demand at any time period

Constraints

$$
\begin{aligned}
& w_{d,s} \leq M \cdot a_s && \forall d,s \\
& \sum_{s,d} \left(A_{s,d,t} \cdot w_{d,s}\right) + u_t \geq R_t && \forall t \\
& o_t = \sum_{s,d} \left(A_{s,d,t} \cdot w_{d,s}\right) + u_t - R_t && \forall t \\
& a_s \in \{0,1\} && \forall s \\
& w_{d,s} \in \mathbb{N}_0 && \forall d,s \\
& o_t, u_t \geq 0 && \forall t
\end{aligned}
$$

Objective

$$\text{minimize } W_1 \sum_t o_t + W_2 \sum_t u_t + W_3 \sum_s a_s \tag{12}$$

A.3 ILP Break Scheduling Single Duty

Sets

Time Periods	$t = \{1,...,Length\}, t^* = \{1,...,Length+1\}$
Breaks	$b = \{1,...,M_{breaks}\}$

Parameters

D_t	The demand for staffing at time period t
M_{breaks}	The maximum number of different breaks+1
Tbt	The total amount of break time required
$Length$	Length of the duty s, in number of time periods
$Bminl$ $(Bmaxl)$	Minimum (maximum) length of a break
$Mlwp$	Minimum long working period
$Lbml$	Long break minimum length
$Wpminl$ $(Wpmaxl)$	Working period minimum (maximum) length
Ebs (Lbs)	Earliest (latest)break start;breaks can start this many time periods after (from) shift start (end)
L	Binary, indicating whether a lunch break is required
$Elbs$ $(Llbs)$	Earliest (Latest) lunch break start
Lbl	Lunch break length
W_1 (W_2)	Penalty cost for overstaffing (understaffing)
M	Used as a sufficiently large constant. Equal to the number of time periods in the duty

Variables

a_b	Binary variable indicating whether the bth break is active
a^l_b	Binary variable indicating whether break b needs to be long
l_b	Binary variable indicating whether break b is the lunch break
b^l_b	The length (in time slots) that the bth break takes
b^s_b	First time slot on which break b is active
wp_b	Indicates the length of the bth working period
$z^s_{t^*,b}$	Binary indicating if the first time period of the bth break is the t^*th time period
$z^e_{t^*,b}$	Binary indicating if the last time period of the bth break is the t^*th time period
$z_{t,b}$	Binary[3] indicating if the duty is on its bth break during time period t
$z^b_{t,b}$	Binary[3] indicating if time period t is before the start of the bth break
$z^a_{t,b}$	Binary[3] indicating if time period t is after the end of the bth break
x_t	Binary[3] indicating if the duty is working during a time period t
u_t, o_t	Understaffing, Overstaffing in time period t

Constraints

$$a_b \leq a_{b-1} \qquad \forall b \in B \setminus \{1\}$$

$$a^l_b + l_b \leq a_b \qquad \forall b$$

$$\sum_b b^l_b = Tbt$$

$$M \cdot (1 - a_b) + b^l_b \geq Bminl \qquad \forall b$$

$$b^l_b \leq Bmaxl \cdot (a_b) + M \cdot (l_b) \qquad \forall b$$

$$a^l_b \cdot Lbml \leq b^l_b \qquad \forall b$$

$$b^s_b \geq Ebs \qquad \forall b$$

[3] These variables can be relaxed to continuous variables on [0,1]. By the binary restriction on $z^s_{t,b}$ and $z^e_{t,b}$ the variables are forced to be either 0 or 1.

$$b_b^s \leq Length - Lbs + (1 - a_b) \cdot (1 + Lbs) \qquad \forall b$$
$$b_b^s \geq (Length + 1) \cdot (1 - a_b) \qquad \forall b$$
$$wp_1 = b_1^s - 1$$
$$wp_b = b_b^s - (b_{b-1}^s + b_{b-1}^l) \qquad \forall b \in B \setminus \{1\}$$
$$wp_b \leq Wpmaxl \qquad \forall b$$
$$wp_1 \geq Wpminl$$
$$M \cdot (1 - a_{b-1}) + wp_b \geq Wpminl \qquad \forall b \in B \setminus \{1\}$$
$$wp_b - Mlwp + 1 \leq a_b^l \cdot M + l_b \cdot M + (1 - a_b) \cdot M \qquad \forall b$$
$$Mlwp - wp_b \leq (1 - a_b^l) \cdot M \qquad \forall b$$
$$\sum_b l_b = L$$
$$(1 - l_b) \cdot M \geq Lbl - b_b^l \qquad \forall b$$
$$(1 - l_b) \cdot M \geq b_b^l - Lbl \qquad \forall b$$
$$(1 - l_b) \cdot M \geq Elbs - b_b^s \qquad \forall b$$
$$(1 - l_b) \cdot M \geq b_b^s - Llbs \qquad \forall b$$
$$\sum_{t^*} z_{t^*,b}^s = 1 \qquad \forall b$$
$$\sum_{t^*} t^* \cdot z_{t,b}^s = b_b^s \qquad \forall b$$
$$\sum_{i=1>Length+1} z_{i,b}^s = z_{t,b}^b \qquad \forall t, b$$
$$\sum_{t^*} z_{t^*,b}^e = 1 \qquad \forall b$$
$$\sum_{t^*} t \cdot z_{t^*,b}^e = b_b^s + b_b^l - 1 + (1 - a_b) \qquad \forall b$$
$$\sum_{i=1<t-1} z_{i,b}^e = z_{t,b}^a \qquad \forall t, b$$
$$z_{t,b} + z_{t,b}^a + z_{t,b}^b = 1 \qquad \forall t, b$$
$$1 - x_t \geq \sum_b z_{t,b} \qquad \forall t \in T$$
$$1 - x_t \geq \sum_b z_{t-1,b}^e \qquad \forall t \in T \setminus \{1\}$$
$$1 - x_1 \leq \sum_b z_{1,b}$$
$$1 - x_t \leq \sum_b z_{t,b} + \sum_b z_{t-1,b}^e \qquad \forall t \in T$$
$$x_t - 1 + u_t \geq D_t \qquad \forall t$$
$$o_t - u_t - x_t + 1 = -D_t \qquad \forall t$$
$$a_b, a_b^l, l_b \in \{0, 1\} \qquad \forall b$$
$$z_{t,b}^s, z_{t,b}^e \in \{0, 1\} \qquad \forall t, b$$
$$o_t, u_t \geq \forall \qquad t$$

Objective

$$\text{minimize } W_1 \sum_t o_t + W_2 \sum_t u_t \tag{13}$$